Continuous integration in an ABAP environment
Statement of Direction

Version 1.3 May 2019
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEGAL DISCLAIMER</td>
<td>3</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>4</td>
</tr>
<tr>
<td>PREREQUISITES</td>
<td>4</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>4</td>
</tr>
<tr>
<td>Basic Idea</td>
<td>4</td>
</tr>
<tr>
<td>Process flow</td>
<td>5</td>
</tr>
<tr>
<td>Our understanding of continuous integration</td>
<td>5</td>
</tr>
<tr>
<td>POTENTIAL USE CASES</td>
<td>7</td>
</tr>
<tr>
<td>WHAT ALREADY EXISTS</td>
<td>10</td>
</tr>
<tr>
<td>In the cloud</td>
<td>10</td>
</tr>
<tr>
<td>WHAT WE PLAN FOR THE INITIAL SHIPMENT</td>
<td>10</td>
</tr>
<tr>
<td>Branch support</td>
<td>10</td>
</tr>
<tr>
<td>Merge support</td>
<td>11</td>
</tr>
<tr>
<td>WHAT WE PLAN FOR LATER RELEASES</td>
<td>11</td>
</tr>
<tr>
<td>EXISTING TOOLS AND PROCESSES</td>
<td>12</td>
</tr>
<tr>
<td>abapGit</td>
<td>12</td>
</tr>
<tr>
<td>Change and Transport System</td>
<td>12</td>
</tr>
<tr>
<td>Pipelines</td>
<td>12</td>
</tr>
<tr>
<td>Lifecycle Management Tools Provided by SAP</td>
<td>12</td>
</tr>
<tr>
<td>SAP Cloud Platform</td>
<td>12</td>
</tr>
<tr>
<td>CI SERVER</td>
<td>13</td>
</tr>
<tr>
<td>WHEN AND WHERE DO WE PLAN TO PUBLISH GIT-ENABLED CTS</td>
<td>13</td>
</tr>
<tr>
<td>In the Cloud</td>
<td>13</td>
</tr>
<tr>
<td>On-Premise</td>
<td>13</td>
</tr>
</tbody>
</table>
LEGAL DISCLAIMER
The information in this document is not subject to your license agreement or any other service or subscription agreement with SAP. Previews or any related presentation and SAP's strategy and possible future developments, products, platforms, directions, and functionality are all subject to change and may be changed by SAP at any time for any reason without notice. The information is not a commitment, promise or legal obligation to deliver any material, code, or functionality. This information is for informational purposes and may not be incorporated into a contract. SAP assumes no responsibility for errors or omissions in the document, except if such damages were caused by SAP’s willful misconduct or gross negligence.

All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as of their dates, and they should not be relied upon in making purchasing decisions.
SUMMARY
This document describes how SAP aims to support continuous integration in an ABAP environment and how this will allow development processes in ABAP to be aligned with development processes that are commonly used for other development environments or languages.
SAP plans to provide options to use state-of-the-art processes and tools for ABAP development and software distribution. This has two main components:
- Git-based versioning, including branch and merge support
- Options to connect your ABAP processes with continuous integration (CI) tools
This will be provided by extending the ABAP Change and Transport System (“CTS”). This new set of CTS features is referred to as “git-enabled CTS” (“gCTS”).

PREREQUISITES
This document requires a basic understanding of the underlying principles of software development in an ABAP system – especially how objects are stored, how changes to objects are handled, and how changes are integrated in other systems (how they are “transported”).

INTRODUCTION
In many development environments, continuous integration (CI) processes are used. They help to manage code changes by multiple developers and bring them together in one integration system as often and early as possible. Developers can begin by working independently on the features they are responsible for. Whenever they reach a state that they think is ready for an integration test, they can promote their development to an environment where the features and fixes implemented by different developers ‘meet’ and can be tested together.

Many developers can easily work in one ABAP system and develop new features. The ABAP development environment ensures that objects are protected or reserved for the developer who is currently working on them. If many developers work in one system, all of their changes will be automatically integrated at a very early point in the development process. But this also means that an object is exclusively locked by the developer who took it first. Therefore, other developers who need to work on the same objects need to coordinate their developments with each other or are forced to work sequentially.

The challenge for SAP is now two-fold: keep the advantages of a central development environment, and make sure that ABAP development processes can be handled similar to the ones which are common on other platforms. This includes, in many development environments, the option to set up new development or testing systems with any selectable state of the coding. This state can be taken from a repository that keeps track of different releases and feature development.

Today, integrating changes into other systems is done via transport requests in ABAP. The import in the receiving systems is usually done by a transport administrator who has corresponding permissions. It is normally not done by the developer. Over time, this has led to some task sharing between developers who release transport requests and transport administrators who maintain import queues (filled with transport requests) and trigger the imports.

You might be used to these processes if you manage ABAP development. This transport process has its pros and cons. It helps a lot in highly regulated environments, but in many cases, it is not flexible enough and does not work with state-of-the-art development processes and tools that are used in CI or DevOps. Especially if you work in a heterogeneous development environment which includes non-ABAP applications, the processes for the different platforms are very different and can hardly be harmonized.

We would like to prepare ABAP for continuous integration and provide options to set up similar development processes in here as in non-ABAP platforms. (Of course, we will not force you to change your existing working mode and processes in ABAP if you do not see an advantage in the CI working mode.)

Basic Idea
The basic idea is that source objects\(^1\) created in an ABAP development environment can be handled in Git:

---

\(^1\) Source objects in here stands for any type of object that can be maintained in an ABAP system.
We enable you to store ABAP source objects, to keep their versions in a Git repository, and thus to create one source of truth for any kind of sources.

The idea is that objects can be transferred to a Git repository and taken back to an environment where they are needed. This could be (another) development environment but also a test or production environment.

Before we start explaining the details of our plans for a Git-enabled CTS (“gCTS”), let’s cover some basic concepts and define some terms.

**Process flow**

The process of moving artifacts from one system to another will work like shown in the following figure:

Any type of ABAP objects shall be enabled to be stored in a Git repository. Whenever objects are changed in a development system, the standard CTS records them in so-called “tasks”. We will leverage these tasks to offer the possibility to transfer and commit these objects to a local repository. From there, these commits are pushed to a remote repository. The objects in the remote repository can then be transferred into the local repository on any target system and from there be imported into ABAP again. Internally the well-known transport tools are used to leverage fast import, automatic object activation and execution of after-import methods.

**Our understanding of continuous integration**

If you look up CI on the Internet, you can find many different definitions and explanations as to what it is. We would therefore like to explain briefly what CI could look in an ABAP environment using gCTS. This will be the basis for the remainder of this document.

The basic flow for continuous integration could look like this:
The steps in this process are:

1. The developer writes code.
   - You will use the known editors.
2. The developer pushes the code changes into a repository.
   - By transferring objects in (open) tasks to Git.
3. CI server monitors the repository.
4. CI server creates a new build:
   - gCTS determines changed objects since last deployment and creates a R3trans data file.
   - This can be done for an existing target system, a new clean one or, a special system that has been set up for developing a certain feature. The build result depends on current state of the target system.
5. CI server deploys the build to the runtime system:
   - The previously created R3trans data file is now deployed into the target system.
6. CI server tests build on the runtime system.
7. CI server sends feedback to the developer.

To manage this process or to automate it, pipelines can be used. An example for a pipeline is shown in the next figure:
A developer writes code in the development environment. Every intermediate development state of changed objects may be committed to a Git repository. We expect commits to be performed on a regular basis, much more often than today’s transport requests are released.

On a CI server, a pipeline is set up to take care of the following steps. It monitors the repository, and whenever a new commit is available, the changed objects are extracted from the repository and deployed to an integration test system. After a successful deployment, some automated tests may run. Feedback on the results of these tests is sent to the developers and the commit is marked as ‘good’.

The pipeline can be set up in a way that, if the test fails, the last ‘good’ commit is deployed again to an integration test system. The integration test system is then always in a usable state and ready for further commits to be deployed. The developers can then do fixes and commit them. Then, the pipeline starts again. This aggregates all of the changes made since the last ‘good’ commit—the newest version of the changed objects—to the integration test system.

With this, the integration test system always shows the last known good software state.

This is a simple example of a pipeline. Many more steps or tasks could be included.

POTENTIAL USE CASES
For using ABAP together with Git, we plan to support the following use cases.

- **Git-based distribution of customer development**: You could then use Git as the technical basis of software distribution. You'll be able to select any software version and deploy it to an ABAP system by selecting a certain commit in the corresponding Git repository. This would, for example, enable rollbacks directly after having imported a certain commit.²

- **Git-based Continuous Integration**

  You will also be able to set up CI pipelines that update the target systems automatically. CI processes involve using a pipeline that defines a sequence of steps to be performed and conditions that control the pipeline.

  The following figure illustrates the systems and servers that are involved and what the process could look like:

---

² We consider the option to roll back changes as a feasible feature in test and quality assurance systems where buggy software shall be reverted immediately.

Reverting changes after productive usage of the changed software raises questions like ‘What about data created with the new software?’, ‘What about actions triggered by the new software?’, and so on. These questions cannot be answered or solved by a simple roll back of a software version and that cannot be solved on a technical level only. This option is therefore not to be seen as a full support for roll-back of any change in a system.
We plan to provide interfaces and services that allow you to set up a pipeline for ABAP development and distribution process. For this to work, you would need an external CI server where the pipeline can be defined. This use case could be completed by using ABAP test systems provided via containers.

The underlying design will follow the process described below in the section *What already exists – in the Cloud.*

- **Maintenance and Feature Development**
  Assume that you have developed your own business application. Now you have decided that it needs to be enhanced with a new feature. At the same time, maintenance of a previous version – which is probably the productive version – has to be supported too.
  In a classic ABAP environment without any Git support, this would mean that you need to set up two system tracks and to do some retrofit or double maintenance to make sure that you can maintain productive code without being forced to publish new features too early and – at the same time – make sure that the maintenance fixes will not be missing in future releases.
  Example: In the following figure, development of new functionality would be done in the system NFD (“new feature development”) and tested in NFQ (“new feature quality assurance”). Fixes for existing functionality would be done in MTD (“maintenance development”) and then be tested in MTQ (“maintenance quality assurance”). MTP is the production system that is used by business where both fixes and new features are made available for productive usage.

Now consider the process with Git in the background: the idea is that you can get rid of the second landscape. You simply create a branch whenever you start to implement a fix or a feature, and you may switch your development system to any of these open branches to continue with the corresponding project. You just choose a certain commit from the maintenance or development branch with which you would like to start a new branch for implementing fixes or new features and import the corresponding state into your single development system.

The productive system will not be affected by these development activities, because it is registered to a productive branch. And you decide how various development branches are consolidated in the master branch and how changes are merged into the productive branch.

The different branches in Git reflect the current state of development.

The following figure shows an example of branches and their possible relations:
Explanation of the example:

- The colored circles symbolize the current commit of each branch. The grey ones are predecessors. The arrows point from a commit to its predecessor(s).
- There’s a master branch, which all fixes and all features are merged into. An integration system may be configured to receive all changes that are committed to the master branch.
- New development is done in a feature branch using an ABAP development system where this feature branch is registered.
- There’s one development system which servers for doing fixes in the maintenance branch and for developing new features in a feature branch – you just switch the branches.
- There’s a production branch into which all fixes from the maintenance branch are merged that shall be deployed into the production system(s). Changes that were done in maintenance branch have to be merged into the master branch, as well (like changes done in the feature branch).

When you finished developing a new feature, you can think of closing the current production branch and starting a new one that starts from the latest commit of the feature branch.

- Distributed Development

Consider the following scenario: Two (teams of) developers would like to create new features or change existing code. They might both need to change the same object. Today, this would mean that one team has to wait until the other team has finishes development and releases the respective objects.

It would be much more efficient if both teams could work in parallel and combine their respective changes later in the cycle.

As shown in the following figure, this would mean that two ‘local’ development environments are needed (so two ABAP runtimes) – one per team and an integration where developments could be combined.

Both teams could now work independently. Whenever they reach an appropriate state, they can push their local development to a feature branch on Git. Then, the feature branch would be merged into the master branch. If, then, the other team published objects that were touched by the first team, mechanisms would be needed to merge these objects to integrate code changes from both teams into the same object.

The latest commit in the master branch is then imported into the integration system.

Details are explained below in the section ‘Merge support’. 
In order to smoothly support distributed development of ABAP artifacts, we will support development teams to organize Git-based development with a set of features:

- Enable developers to define which packages are maintained in which repository – thus supporting development teams to organize their software components in repositories;
- Integration in standard workbench organizer: keep changes separate from each if touched objects are assigned to different repositories or none;
- Enable developers to transfer objects directly from editing environment into their assigned repository (no release of transports necessary);
- Allow development teams to propagate their changes by leveraging repository functionality (e.g. merging feature branches to master branch) instead of releasing transports.

**WHAT ALREADY EXISTS**

Some parts of the Git-enabled CTS are already implemented and are in use. In this section, we will explain how this functionality is already being used. And we’ll explain how we plan to interact with other offerings of SAP that have or might have connections to a Git-enabled CTS.

**In the cloud**

SAP Cloud Platform ABAP Environment uses a Git-enabled CTS under the hood when it comes to distributing customer development artifacts from the development system to another system (testing or productive, for example). The ABAP objects are transferred to follow-on systems with the help of Git. These steps happen in the background. They are not visible to the user and don’t require any configuration.

Check the blog by Harald Kuck to learn more about SAP Cloud Platform ABAP Environment in general: https://blogs.sap.com/2018/09/04/sap-cloud-platform-abap-environment

**WHAT WE PLAN FOR THE INITIAL SHIPMENT**

To support at least the use cases Git-based software distribution and Git-based continuous integration as outlined above, two main features will be available: branch support and merge support. We plan to support this as follows:

**Branch support**

Branches are needed to be able to provide usable and tested software for all systems. Anything that has been developed in a development branch can be merged into a test branch and then be activated on the test system. If subsequent testing was successful, the same software could then be moved to the production system in the same manner.

In the following section, we provide a more detailed description on what is planned concerning merge support.
Using different branches would allow the setup of a pipeline on a CI server which could help automate your processes.

**Merge support**

**Basic merge support**
For the first version, we plan to start with a simple level of merging:
- You trigger merging from within gCTS. To support this, we plan to offer a configuration option where you can decide which branch shall win (‘yours’ – meaning the version contained in the current branch as latest commit, or ‘theirs’ – meaning the new objects that are asking to become part of the branch that you are working on).
- You can use the tools offered by Git to merge two files. This is a generic, text-based merge.

**WHAT WE PLAN FOR LATER RELEASES**

**Improved merge support**
To improve support for use cases ‘Maintenance and Feature Development’ and ‘Distributed Development’, conflict resolution must be as convenient as possible.
To support you better with working in different branches, we plan to improve merging:
- We think about introducing human-readable object formats for the most common ABAP object types so that the representation in a repository is similar to what is shown in the editors. With this, text-based conflict resolution becomes simpler.
- For objects without human-readable format, we plan to provide a generic merge tool for objects that do not support the common object format that makes sure that the results of conflict resolution are compatible with the gCTS file format of this object type.

**Assignment of packages to repositories and workbench organizer integration**
To support users to organize distributed development, we plan to offer the features that are outlined in section “distributed development”.

**Customizing data**
We plan to cover customizing, too. This shall include more than simple table entries.

**Dependencies**
We plan to offer the possibility that repositories describe themselves as components, and to refer to certain component versions as prerequisites. Thus, it will be possible to manage dependencies between component versions and to ensure consistency during deployment.
EXISTING TOOLS AND PROCESSES
In this section, you will find information about how we plan to work with tools that already exist in the market but might have interferences with a Git-enabled CTS.

abapGit
abapGit (see https://docs.abapgit.org for detailed documentation) is an open source project driven by the SAP community since 2014. It partly covers the same use cases as gCTS but focuses on using Git repositories to quickly share ABAP code between arbitrary systems that are typically not connected through a transport landscape. AbapGit is available for SAP NetWeaver AS ABAP as of release 7.02. In the context of SAP Cloud Platform ABAP Environment, abapGit can be used to import demo content or to initially transfer ABAP development objects from on-premise ABAP systems into an SAP Cloud Platform.

Git works on files. That means ABAP objects must be serialized to the file system. There are currently different file formats used by abapGit and gCTS. On the one hand, while gCTS supports all transportable ABAP object types with a generic xml-based file format, abapGit handles most common object types of ABAP workbench with object type specific handlers (see https://docs.abapgit.org/ref-supported.html). On the other hand, the current generic file format of gCTS is much less suited for peer code reviews on Git than the custom format of abapGit. For this reason, an initiative is driven by SAP to define a common file format that fulfills the following requirements:

- complete (can be used for distribution of software),
- aligned (between abapGit and gCTS),
- compatible (between releases, so that objects that have been pushed to Git in an older release can be pulled into a newer),
- human-readable (usable for code reviews, decisions on "cherry picking", etc.).

It is important to note the different support level implied by abapGit’s open source license. The code is open, reproducible, and a lively community usually provides fixes in acceptable time but without any liability or warranty. While SAP started contributing to the project in 2018, SAP offers no official support for it.

Change and Transport System
All of what has been described in this document up to now is planned new additional functionality. Existing functionality and processes will not be changed or deprecated. Anything that you have set up using the Change and Transport System (CTS) in on-premise systems, the enhanced Change and Transport System (CTS+), or Change Request Management and Quality Gate Management of SAP Solution Manager (based on central CTS) will continue working as it does today.

Git-enabled CTS is intended to be an additional option. You may start using it in a pilot or testing project for some of your packages. Switching on Git-enabled CTS will not harm your existing landscape and projects. You can use gCTS e.g. for your development, integration and test system. We plan to offer options to switch to classical CTS after the deployment and testing in test system was successful.

Pipelines
SAP provides some sample pipelines at https://sap.github.io/jenkins-library/. We plan to provide sample pipelines using gCTS, as well.

Lifecycle Management Tools Provided by SAP
On SAP Solution Manager, SAP provides compliance to change management standards such as ITIL, namely Change Request Management (ChaRM) and Quality Gate Management (QGM). SAP also plans to support use cases in the future that will allow using CI processes in development without losing compliance capabilities. The same applies to SAP’s upcoming cloud-based solution, Cloud Application Lifecycle Management (CALM).

SAP Cloud Platform
In SAP Cloud Platform, you can develop using various technologies. For these non-ABAP technologies, you can use CI processes that are described on the Internet and especially in a best practice guide on developers.sap.com:
With **gCTS** we intend to provide service interfaces that enable you to harmonize the change processes on SAP Cloud Platform with ABAP systems (on-premise and in the cloud).

**CI SERVER**
You can find several providers of CI servers in the market. Jenkins is a prominent example.
We plan to offer public interfaces that you can use to set up a CI process with a CI Server of your choice.
It might be that at some point in time, we publish sample pipelines for commonly used CI servers.

**WHEN AND WHERE DO WE PLAN TO PUBLISH GIT-ENABLED CTS**

**In the Cloud**
New functionality is planned to be shipped with SAP Cloud Platform ABAP Environment in different releases.

**On-Premise**
We plan to provide a first subset of features described in this document with the next SAP S/4 HANA release at the end of 2019.